

Express Mailing Label No.: EU070956742US

Docket No.: 1027.2.1

**UNITED STATES PATENT APPLICATION**

of

David B. Buehler

for

**RECURSIVE RAY CASTING METHOD AND APPARATUS**

20030220-02998001

# RECURSIVE RAY CASTING METHOD AND APPARATUS

## BACKGROUND OF THE INVENTION

### 1. The Field of the Invention

The present invention relates generally to graphical rendering devices and systems. Specifically, the invention relates to devices and systems for conducting highly realistic three-dimensional graphical renderings.

### 2. The Relevant Art

Graphical rendering involves the conversion of one or more object descriptions to a set of pixels that are displayed on an output device such as a video display or image printer. Object descriptions are generally mathematical representations that model or represent the shape and surface characteristics of the displayed objects. Graphical object descriptions may be created by sampling real world objects and/or by creating computer-generated objects using various editors.

In geometric terms, rendering requires representing or capturing the details of graphical objects from the viewer's perspective to create a two-dimensional scene or projection representing the viewer's perspective in three-dimensional space. The two-dimensional rendering facilitates viewing the scene on a display device or means such as a video monitor or printed page.

A primary objective of object modeling and graphical rendering is realism, i.e., a visually realistic representation that is life-like. Many factors impact realism, including surface detail, lighting effects, display resolution, display rate, and the like. Due to the complexity of real-world scenes, graphical rendering systems are known to have an insatiable thirst for processing power and data throughput. Currently available rendering systems lack the performance necessary to make photo-realistic renderings in real-time.

1  
2 To increase rendering quality and reduce storage requirements, surface details are  
3 often separated from the object shape and are mapped onto the surfaces of the object  
4 during rendering. The object descriptions including surface details are typically stored  
5 digitally within a computer memory or storage medium and referenced when needed.

6 One common method of representing three-dimensional objects involves  
7 combining simple graphical objects into a more realistic composite model or object. The  
8 simple graphical objects, from which composite objects are built, are often referred to as  
9 primitives. Examples of primitives include triangles, surface patches such as bezier  
10 patches, and voxels.

11 Voxels are volume elements, typically cubic in shape, that represent a finite,  
12 three-dimensional space similar to bitmaps in two-dimensional space. Three-dimensional  
13 objects may be represented using a primitive comprising a three-dimensional array of  
14 voxels. A voxel object is created by assigning a color and a surface normal to certain  
15 voxel locations within the voxel array while marking other locations as transparent.

16 Voxel objects reduce the geometry bandwidth and processing requirements  
17 associated with rendering. For example, objects represented with voxels typically have  
18 smaller geometry transform requirements than similar objects constructed from triangles.

19 Despite this advantage, existing voxel rendering algorithms are typically complex and  
20 extremely hardware intensive. A fast algorithm for rendering voxel objects with low  
21 hardware requirements would reduce the geometry processing and geometry bandwidth  
22 requirements of rendering by allowing certain objects to be represented by voxel objects  
23 instead of many small triangles.

24 As mentioned, rendering involves creating a two-dimensional projection  
25 representing the viewer's perspective in a three-dimensional space. One common method  
26 of creating a two-dimensional projection involves performing a geometric transform on  
27

1 the primitives that comprise the various graphical objects within a scene. Performing a  
2 geometric transform changes any coordinates representing objects from an abstract space  
3 known as a world space into actual device coordinates such as screen coordinates.

4 After a primitive such as a triangle has been transformed to a device coordinate  
5 system, pixels are generated for each pixel location which is covered by that primitive.  
6 The process of converting graphical objects to pixels is sometimes referred to as  
7 rasterization or pixelization. Texture information may be accessed in conjunction with  
8 pixelization to determine the color of each of the pixels. Because more than one  
9 primitive may be covering any given location, a z-depth for each pixel generated is also  
10 calculated, and is used to determine which pixels are visible to the viewer.

11 Figures 1a and 1b depict a simplified example of graphical rendering. Referring to  
12 Figure 1a, a graphical object 100 may be rendered by sampling attributes such as object  
13 color, texture, and reflectivity at discrete points on the object. The sampled points  
14 correspond to device-oriented regions, typically round or rectangular in shape, known as  
15 pixels 102. The distance between the sampled points is referred to herein as a sampling  
16 interval 104. The sampled attributes, along with surface orientation (*i.e.* a surface  
17 normal), are used to compute a rendered color 108 for each pixel 102. The rendered  
18 colors 108 of the pixels 102 preferably represent what a perspective viewer 106 would  
19 see from a particular distance and orientation relative to the graphical object 100.

20 As mentioned, the attributes collected by sampling the graphical object 100 are  
21 used to compute the rendered color 108 for each pixel 102. The rendered color 108  
22 differs from the object color due to shading, lighting, and other effects that change what  
23 is seen from the perspective of the viewer 106. The rendered color 108 may also be  
24 constrained by the selected rendering device. The rendered color may be represented by a  
25 set of numbers 110 designating the intensity of each of the component colors of the  
26 selected rendering device, such as red, green, and blue on a video display or cyan,  
27 magenta, yellow, and black on an inkjet printer.

1 As the graphical object 100 is rendered with each frame, the positioning and  
2 spacing of the discrete sampling points (i.e., the pixels 102) projected onto the graphical  
3 object 100 determine what is seen by the perspective viewer 106. One method of  
4 rendering, referred to as ray tracing, involves determining the position of the discrete  
5 sampling points by extending a grid 111 of rays 112 from a focal point 114 to find the  
6 closest primitive each ray intersects. Since the rays 112 are diverging, the spacing  
7 between the rays 112, and therefore the size of the grid 111, increases with increasing  
8 distance. Ray tracing, while precise and accurate, is generally not used in real-time  
9 rendering systems due to the computational complexity of currently available ray tracing  
10 algorithms.

11 The grid 111, depicted in Figure 1a, is a set of regularly spaced points  
12 corresponding to the pixels 102. The points of the grid 111 lie in an image plane  
13 perpendicular to a ray axis 115. The distance of each pixel 102 from a reference plane  
14 perpendicular to the ray axis 115, such as the grid 111, is known as the pixel depth or z-  
15 depth. The distance or depth of the graphical object 100 changes the level of detail seen by  
16 the perspective viewer 106. Relatively distant objects cover a smaller rendering area on the  
17 display device, resulting in a reduced number of rays 112 that reach the graphical object 100,  
18 and an increased sampling interval 104.

19 Visual artifacts occur when the spacing between the rays 112 result in the  
20 sampling interval 104 being too large to faithfully capture the details of the graphical  
21 object 100. A number of methods have been developed to eliminate visual artifacts  
22 related to large sampling intervals. One method, known as super-sampling, involves  
23 rendering the scene at a higher resolution than the resolution used by the output device,  
24 followed by a smoothing or averaging operation to combine multiple rendered pixels into  
25 a single output pixel.

26 Another method, developed to represent objects at various distances and sampling  
27 intervals faithfully, involves creating multiple models of a given object. Less detailed

models are used when an object is distant, while more detailed models are used when an object is close. Texture information may also be stored at multiple resolutions. During rendering, the texture map appropriate for the distance from the viewer is utilized.

The graphical objects, and portions thereof, that are visible to a viewer are dependent upon the perspective of the viewer. Referring to Figure 1b, a graphical scene 150 may include a variety of the graphical objects 100, some of which may be visible while others may be obstructed. Unobstructed objects are often designated as foreground objects 100a, while partially obstructed objects may be referred to as background objects 100b. Within the graphical scene 150, completely obstructed objects may be referred to as non-visible objects.

During rendering, the graphical scene 150 is converted to rendered pixels on a rendering device for observance by an actual viewer. Each rendered pixel preferably contains the rendered color 108 such that the actual viewer's visual perception of each graphical object 100 is that of the perspective viewer 106.

A small percentage of the graphical objects 100 may be visible within a particular graphical scene. For example, the room shown within the graphical scene 150 may be one of many rooms within a database containing an entire virtual house. The rendering of non-visible objects and pixels unnecessarily consumes resources such as processing cycles, memory bandwidth, memory storage, and function specific circuitry. Since the relative relationship of graphical objects changes with differing perspectives, for example as the perspective viewer 106 walks through a virtual house, the ability to dynamically determine and prune non-visible objects and pixels improves rendering performance.

Ray casting is a method to determine visible objects and pixels within a graphical scene 150 as shown in Figure 1a. Ray casting is one method of conducting ray tracing that advances (casts) one ray for each pixel within the graphical scene 150 from the perspective viewer 106. With each cast one or more graphical objects are tested against

each ray to see if the ray has “collided” with the object – an extremely processing-intensive procedure.

Z-buffering is another method that is used to determine visible pixels. Pixels are generated from each potentially visible object and stored within a z-buffer. A z-buffer typically stores a depth value and a pixel color value at a memory location corresponding to each x,y position within the graphical scene 150. A pixel color value is overwritten with a new value only if the new pixel depth is less than the depth of the currently stored pixel.

Referring to Figure 2, a method of rendering known as post z-buffer shading and texturing defers shading and texturing operations within a rendering pipeline 200 and therefore does not texture or shade non-visible pixels. In a typical rendering system, the color of the pixels is calculated prior to z-buffering. In a post z-buffer shading and texturing system, such as the rendering pipeline 200, final color calculations are not performed until after the z-buffering operation. Deferred shading and texturing eliminates the memory lookups and processing operations associated with shading and texturing non-visible pixels and thereby facilitates increased system efficiency.

The rendering pipeline 200 includes a display memory 210 and a graphics engine 220 comprised of a triangle converter 230, a z-buffer 240, and a shading and texturing engine 250. The rendering pipeline 200 also includes a frame buffer 260. In the depicted embodiment, the display memory 210 receives and provides various object descriptors 212 that describe the graphical objects 100.

The display memory 210 preferably contains descriptions of those objects that are potentially visible in the graphical scene 150. With scene changes, the object descriptors 212 may be added or removed from the display memory 210. In some embodiments, the display memory 210 contains a database of the object descriptors 212, for example, a database describing an entire virtual house.

1           Some amount of simple pruning may be conducted on objects within the display  
2 memory 210, for example, by software running on a host processor. Simple pruning may  
3 be conducted so that the graphical objects that are easily identified as non-visible are  
4 omitted from the rendering process. For example, those graphical objects 100 that are  
5 completely behind the perspective viewer 106 may be omitted or removed from the  
6 display memory 210.

7           The graphics engine 220 retrieves the object descriptors 212 from the display  
8 memory 210 and presents them to the triangle converter 230. In the depicted  
9 embodiment, the object descriptors 212 define the vertices of a triangle or set of triangles  
10 and their associated attributes such as the object color. Typically, these attributes are  
11 interpolated across the face of the triangle to provide a set of potentially visible pixels  
12 232.

13           The potentially visible pixels 232 are received by the z-buffer 240 and processed  
14 in the manner previously described to provide the visible pixels 242 to the shading and  
15 texturing engine 250. The shading and texturing engine 250 textures and/or shades the  
16 visible pixels 242 to provide rendered pixels 252 that are collected by the frame buffer  
17 260 to provide one frame of pixels 262. The framed pixels 262 are typically sent to a  
18 display system for viewing.

19           One difficulty in conducting post z-buffer shading and texturing is the increased  
20 complexity required of the z-buffer. The z-buffer is required to contain additional  
21 information relevant to shading and texturing in addition to the pixel depth. The z-buffer  
22 is often a performance critical element, in that each pixel is potentially updated multiple  
23 times, requiring increased bandwidth. The increased size and bandwidth requirements on  
24 the z-buffer have limited the use of post z-buffer shading and texturing within graphical  
25 systems.

26           One prior art method to reduce the size of the z-buffer is shown in Figure 3. The  
27 method divides a screen 300 into tiles 310. The tiles 310 and the screen 300 consist of a



1 plurality of scanlines 320. Each tile 310 is rendered as if it were the entire screen 300,  
2 thus requiring a tile-sized z-buffer. While a tile-sized z-buffer requires less memory, a  
3 tile-sized z-buffer increases complexity related to sorting, storing, accessing, and  
4 rendering the object descriptors 212 within the display memory 210. The increased  
5 complexity results from objects that overlap more than one tile.

6 While many advances have been made to graphical rendering algorithms and  
7 architectures, including those depicted in the graphical pipeline 200, real-time rendering  
8 of photo-realistic life-like scenes requires the ability to render greater geometric detail  
9 than is sustainable on currently available graphical rendering systems.

10 Therefore, what is generally needed are methods and apparatus to conduct  
11 efficient graphical rendering. Specifically, what is needed is a graphical system that  
12 renders voxel primitives efficiently. The ability to render voxel objects efficiently  
13 increases the detail achievable in real-time graphical rendering systems.

14 What is also needed is a graphical system that renders very detailed scenes with  
15 extensive depth complexity, without tying up external memory interfaces with z-buffer  
16 data traffic. A z-buffering apparatus and method that facilitates large tiles, supports a  
17 high pixel throughput, is compact enough to reside entirely on-chip, and reduces external  
18 memory bandwidth requirements would facilitate such a system.

19 In addition to better z-buffering, a method and apparatus are needed that reduce  
20 the bandwidth load on the z-buffer. Specifically, what is needed is a method and  
21 apparatus that reduces the generation of non-visible pixels prior to z-buffering.

22 In addition to more intelligent pixel generation, rendering highly realistic scenes  
23 requires accessing large amounts of texture and world description data. Specifically,  
24 what is needed is an apparatus and method to maximize the efficiency of internal and  
25 external memory accesses. Such a method and apparatus would preferably achieve  
26 increased realism by facilitating larger stores of texture data within low-cost external  
27 memories, while maintaining a high data throughput within the rendering pipeline.

1           Lastly, what is needed is a graphical processing architecture that facilitates  
2 combining the various elements of the present invention into an efficient rendering  
3 pipeline that is scalable in performance.  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27

EXHIBIT 100-999999

BRIAN C. KUNZLER  
ATTORNEY AT LAW  
10 WEST 100 SOUTH, SUITE 425  
SALT LAKE CITY, UTAH 84101

## OBJECTS AND BRIEF SUMMARY OF THE INVENTION

The apparatus of the present invention has been developed in response to the present state of the art, and in particular, in response to the problems and needs in the art that have not yet been fully solved by currently available graphical rendering systems and methods. Accordingly, it is an overall object of the present invention to provide an improved method and apparatus for graphic rendering that overcomes many or all of the above-discussed shortcomings in the art.

To achieve the foregoing objects, and in accordance with the invention as embodied and broadly described herein in the preferred embodiments, an apparatus and method for improved graphical rendering is described. The apparatus and method facilitate increased rendering realism by supporting greater geometric detail, efficient voxel rendering, larger amounts of usable texture data, higher pixel resolutions including super-sampled resolutions, increased frame rates, and the like.

In a first aspect of the invention, a method and apparatus for casting ray bundles is described that casts entire bundles of rays relatively large distances. The ray bundles are subdivided into smaller bundles and casting distances as the rays and bundles approach a graphical object. Each bundle advances in response to a single test that is conducted against a proximity mask corresponding to a particular proximity. Sharing a single proximity test among all the rays within a bundle greatly reduces the processing burden associated with ray tracing. Individual rays are generated when a ray bundle is within close proximity to the object being rendered. The method and apparatus for casting ray bundles efficiently calculates the first ray intersections with an object and is particularly useful for voxel objects.

In a second aspect of the invention, a method and apparatus for gated pixelization (*i.e.*, selective pixel generation) is described that conducts z-buffering at a coarse depth resolution using minimum and maximum depths for a pixel set. In one embodiment, the method and apparatus for gated pixelization maximizes the utility of reduced depth

1 resolution by shifting the range of depths stored within the z-buffer in coordination with  
2 the depth of the primitives being processed. The method and apparatus for gated  
3 pixelization also reduces the bandwidth and storage burden on the z-buffer and increases  
4 the throughput of the pixel generators.

5 In a third aspect of the invention, a method and apparatus for z-buffering pixels is  
6 described that stores and sorts the pixels from an area of the screen, such as a tile, into  
7 relatively small regions, each of which is processed to determine the visible pixels in each  
8 region. The method and apparatus facilitates high throughput z-buffering, efficient  
9 storage of pixel auxiliary data, as well as deferred pixel shading and texturing.

10 In a fourth aspect of the invention, an apparatus and method for sorting memory  
11 accesses related to graphical objects is described that increases the locality of memory  
12 references and thereby increases memory throughput. In the presently preferred  
13 embodiment, access requests for a region of the screen are sorted and stored according to  
14 address, then accessed page by page to minimize the number of page loads that occur.  
15 Minimizing page loads maximizes the utilization of available bandwidth of graphical  
16 memory interfaces.

17 The various aspects of the invention are combined in a pipelined graphics engine  
18 designed as a core of a graphics subsystem. In the presently preferred embodiment,  
19 graphical rendering is tile-based and the pipelined graphics engine is configured to  
20 efficiently conduct tile-base rendering.

21 The graphics engine includes a set of pixel generators that operate in conjunction  
22 with one or more occlusion detectors. The pixel generators include voxel ray tracers,  
23 which use the method and apparatus for casting ray bundles to greatly reduce the number  
24 of computations required to determine visible voxels. In the preferred embodiment, the  
25 voxel objects are stored and processed in a compressed format.

26 The voxel ray tracers generate pixels from voxel objects by calculating ray  
27 collisions for the voxel objects being rendered. Proximity masks are preferably generated

1 previous to pixel generation. Each proximity mask indicates the voxel locations that are  
2 within a certain distance of a nontransparent voxel. The proximity masks are brought in  
3 from external memory and cached as needed during the rendering process. An address  
4 that references the color of the particular voxel impinged upon by each ray is also  
5 calculated and stored within a pixel descriptor.

6 The voxel ray tracers conduct ray bundle casting to efficiently determine any first  
7 ray intersections with a particular voxel object. The voxel ray tracers are preferably  
8 configured to conduct perspective ray tracing where the rays diverge with each cast.

9 Ray tracing commences by initializing the direction of the rays in the voxel  
10 object's coordinate system, based on the voxel object's orientation in world space and the  
11 location of the viewer. The casting direction of each ray bundled is represented by a  
12 single directional vector. A bundle width and height corresponding to a screen region  
13 represent the bundle size. In the preferred embodiment, a top level bundle may comprise  
14 100 or more rays.

15 Each ray bundle is advanced by casting the bundle in the direction specified by  
16 the directional vector a selected casting distance. A proximity mask is selected for testing  
17 that preferably indicates a proximity to the object surface that corresponds with the  
18 selected casting distance. The single test against the properly selected proximity mask  
19 ensures that none of the rays in a bundle could have intersected the object between the  
20 last test and the current test.

21 A positive proximity test indicates that at least one ray is within a certain distance  
22 of the object surface. In response to a positive proximity test, the ray bundle is preferably  
23 subdivided into smaller bundles that are individually advanced, tested, and subdivided  
24 until each bundle is an individual ray. The individual rays are also advanced and tested  
25 against a collision mask that indicates impingement of the ray on a non-transparent voxel  
26 of the object of interest. Upon impingement, a color lookup address for the impinged  
27 voxel is calculated, and stored along with x and y coordinates in the pixel descriptor.

The method and apparatus for casting ray bundles has several advantages and is particularly useful for voxel objects. Casting is very efficient, in that the majority of tests performed (for each ray that intersects the surface) are shared by many other rays within each bundle the ray was a member of. The proximity mask information is compact, particularly when compressed, and may be cached on-chip for increased efficiency. The algorithm is also memory friendly, in that only those portions of the object that are potentially visible need be brought onto the chip *i.e.* efficiency is maintained with partial view rendering. Perhaps the greatest advantage, particularly when conducted in conjunction with voxel objects, is a substantial reduction in the number of, and the bandwidth required for, geometry calculations within highly detailed scenes. The recursive subdividing nature of the algorithm also facilitates parallel execution, which in certain embodiments facilitates computing multiple ray intersections per compute cycle.

The pixel generators, such as the voxel ray tracers, generate potentially visible pixels, working in conjunction with the occlusion detector. The occlusion detector conducts depth checking at a coarse depth resolution in order to gate the pixel generators, thereby allowing the pixel generators to skip generating pixels for locations known to be occluded by a previously processed pixel. The preferred embodiment of the occlusion detector performs a parallel comparison of all the depth values within a region to a given value, and returns a mask indicating the pixel locations that are occluded at that depth. The pixel generators use the mask information to generate only pixels that are not known to be occluded. Using the occlusion detectors to conduct pixel gating reduces the overall processing and storage burden on the z-buffer.

In the preferred embodiment, the occlusion detector is used in conjunction with front-to-back rendering of the graphical primitives that comprise a scene. In certain embodiments, the occlusion detector is capable of shifting the depth range in which occlusions are detected. Depth shifting focuses the available resolution of the occlusion detector on a limited depth range. Depth shifting is preferably conducted in conjunction

The pixel generators and the occlusion detectors coordinate to conduct gated pixelization and provide potentially visible pixels to a sorting z-buffer. The sorting z-buffer includes a region sorter, a region memory, and a region-sized z-buffer. The region sorter sorts the potentially visible pixels according to their x,y coordinates within a screen or tile to provide sorted pixels. The sorted pixels corresponding to each region within a graphical scene or tile are received and processed by a region-sized z-buffer to provide the visible pixels.

Sorting the pixels into regions facilitates the use of a very small z-buffer at the core of the sorting z-buffer. The screen regions corresponding to the region-sized z-buffer are preferably smaller than the tiles typical of rendering systems. Sorting the pixels into regions also facilitates the use of larger tiles. Larger tiles reduce the number of graphic primitives that overlap more than one tile.

In the preferred embodiment, the bucket sorter stores the received pixels by conducting a parallel transfer to the region memory. Since the pixels may originate from the

1 same primitive, the received pixels often have a certain amount of spatial coherence. In the  
2 preferred embodiment, the bucket sorter exploits spatial coherence by conducting a first level  
3 of bucket sorting as the pixels arrive. Additional levels of bucket sorting may be performed  
4 by recursively processing the contents of the region memory.

5 A further stage of the sorting z-buffer is the pixel combiner. The pixel combiner  
6 monitors the pixels provided by the sorting z-buffer. In those instances where super-  
7 sampled anti-aliasing is performed, combining is conducted on those pixels that can be  
8 combined without loss of visual quality. Combining is preferred for super-sampled pixels  
9 combined without loss of visual quality. Combining is preferred for super-sampled pixels  
10 that reference the same texture. Combining reduces the load on the colorization engine  
11 and the anti-aliasing filter.

12 The sorting z-buffer provides visible pixels to a colorization engine. The  
13 colorization engine colorizes the pixels to provide colorized pixels. In the present  
14 invention, colorizing may comprise any operation that affects the rendered color of a  
15 pixel. In one embodiment, the colorizing of pixels includes shading, texturing, normal  
16 perturbation (*i.e.* bump mapping), as well as environmental reflectance mapping.  
17 Colorizing only those pixels that are visible reduces the processing load on the  
18 colorization engine and reduces the bandwidth demands on external texture memory.

19 The colorization engine colorizes pixels using a set of pixel colorizers, an attribute  
20 request sorter, and a set of attribute request queues. The graphics engine may also  
21 include or be connected to a pixel attribute memory containing pixel attributes that are  
22 accessed by the pixel colorizers in conjunction with colorization. Voxel color data is  
23 preferably stored in a packed array so that only nontransparent voxels on the surface of an  
24 object need be stored. Surface normal information is also stored along with the color.

25 The attribute request sorter routes and directs the attribute requests relevant to  
26 pixel colorization to the various attribute request queues. In one embodiment, the  
27 attribute request sorter sorts the attribute requests according to the memory page in which



the requested attribute is stored, and the attribute request sorter routes the sorted requests to the pixel attribute memory.

Sorting the attribute requests increases the performance and/or facilitates the use of lower cost storage by increasing the locality of memory references. In one embodiment, increasing the locality of memory references facilitates using greater quantities of slower, less costly dynamic random access memory (DRAM) within a memory subsystem while maintaining equivalent data throughput.

In the preferred embodiment, the last portion in the pipeline is the anti-aliasing filter. In those instances where super-sampling is performed, multiple super-sampled pixels are combined to provide rendered pixels. The rendered pixels are stored in the frame buffer and used to provide a high quality graphical rendering.

The various elements of the graphics engine work together to accomplish high performance, highly detailed rendering using reduced system resources. Pixel descriptors are judiciously generated in the pixelizers by conducting gated pixelization. Each pixel descriptor, though grouped with other pixels of the same screen region, flows independently through the various pipeline stages. Within each pipeline stage, the number of processing units operating in parallel is preferably scalable in that each pixel is directed to an available processing unit.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

In order that the manner in which the advantages and objects of the invention are obtained will be readily understood, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof which are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered to be limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

Figure 1a is partially schematic respective view depicting a prior art method of rendering a graphical object;

Figure 1b is a perspective view of a graphical scene in accordance with graphical rendering systems;

Figure 2 is a schematic block diagram depicting a prior art graphics pipeline;

Figure 3 is a chart depicting a prior art tile-based rendering method;

Figure 4a is a schematic block diagram depicting one embodiment of a graphical rendering system in accordance with the invention;

Figure 4b is a schematic block diagram depicting one embodiment of a graphics subsystem in accordance with the present invention;

Figure 5 is a schematic block diagram depicting one embodiment of a graphical rendering apparatus of the present invention;

Figure 6 is a schematic block diagram depicting one embodiment of a graphical rendering method of the present invention;

Figure 7 is a schematic block diagram depicting one embodiment of a pixel generation apparatus of the present invention;

Figure 8a is a schematic block diagram depicting one embodiment of a triangle pixelization apparatus of the present invention;

Figure 18a is a schematic block diagram depicting one embodiment of a bucket sorting apparatus of the present invention;

1 Figure 18b is a schematic block diagram depicting an on-chip embodiment of a  
2 bucket sorting apparatus of the present invention;

3 Figure 19 is a flow chart diagram depicting one embodiment of a bucket sorting  
4 method of the present invention;

5 Figure 20a is a schematic block diagram depicting one embodiment of a sorting z-  
6 buffer apparatus of the present invention;

7 Figure 20b is a flow chart diagram depicting one embodiment of a sorting z-buffer  
8 method of the present invention;

9 Figure 21a is a schematic block diagram depicting one embodiment of a graphics  
10 memory localization apparatus of the present invention;

11 Figure 21b is a flow chart diagram depicting one embodiment of a graphics  
12 memory localization method of the present invention;

13 Figure 22 is a schematic block diagram depicting one embodiment of a pixel  
14 colorization apparatus of the present invention; and

15 Figure 23 is a flow chart diagram depicting one embodiment of a pixel  
16 colorization method of the present invention.

17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27

2022-02-22

BRIAN C. KUNZLER  
ATTORNEY AT LAW  
10 WEST 100 SOUTH, SUITE 425  
SALT LAKE CITY, UTAH 84101

**DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS**

Referring to Figure 4a, a digital media system 400 in accordance with the present invention may include a CPU 410, a storage device 420, a memory 430, an audio subsystem 440, and a graphics subsystem 450, interconnected by a system bus 412. In addition, the graphical rendering system 400 may include speakers 445 and a video display 455. In the depicted embodiment, the speakers 445 receive and play an audio signal 442 from the audio subsystem 440, while the video display 455 receives and displays a video signal 452 from the graphics subsystem 450. The digital media system 400 may be a multimedia system such as a game console or personal computer.

Referring to Figure 4b, one embodiment of the graphics subsystem 450 in accordance of the present invention includes a transform engine 460, a display memory 470, a graphics engine 480, and a frame buffer 490. The transform engine 460 receives data such as the object descriptors 212 from the system bus 412. In the preferred embodiment, the transform engine 460 converts the coordinates associated with the object descriptors 212 into screen coordinates such as those seen by the perspective viewer 106. The display memory 470 stores the object descriptors 212 and provides them to the graphics engine 480.

The graphics engine 480 converts the object descriptors 212 to rendered pixels 482, while the frame buffer 490 and associated circuitry converts the rendered pixels 482 to the video signal 452. In one embodiment, the display memory 470 is substantially identical to the (prior art) display memory 210 and the frame buffer 490 is substantially identical to the (prior art) frame buffer 260.

Figure 5 is a schematic block diagram depicting one embodiment of the graphics engine 480 of the present invention. The graphics engine 480 may be embodied in hardware, software or a combination of the two. In the preferred embodiment, the graphics engine 480 is pipelined, operating on batches of pixels corresponding to a single tile. For example, the sorting z-buffer may operate on objects or pixels corresponding to a first tile, while the

1 colorizing engine works on pixels corresponding to a second tile. When the colorizing  
2 engine has finished colorizing the pixels, the pixels are sorted into screen order and  
3 antialiased, generating rendered pixels.

4 In the depicted embodiment, the graphics engine 480 includes a set of pixel  
5 generators 510 that operate in conjunction with one or more occlusion detectors 520 to  
6 conduct gated pixelization. The pixel generators 510 receive the object descriptors 212  
7 and provide potentially visible pixels 512 to a sorting z-buffer 530. The occlusion  
8 detectors 520 gate the pixelization conducted by the pixel generators by maintaining a  
9 current occlusion depth for each pixel position.

10 As shown in Figure 4, the object descriptors 212 may be provided by the display  
11 memory 470. The object descriptors 212 describe graphical objects, such as the graphical  
12 object 100 of figure 1. Each object may be composed of multiple sub-objects or  
13 primitives such as triangles, bezier patches, and voxel arrays. In the preferred  
14 embodiment, each sub-object corresponds to one object descriptor 212 resulting in  
15 multiple object descriptors 212 for those objects that are composed of multiple sub-  
16 objects.

17 Processing is preferably conducted on each object descriptor 212 independent of  
18 other object descriptors. For purposes of clarity, the description of this invention  
19 typically implies a single object descriptor 212 for each graphical object 100, though  
20 multiple object descriptors 212 are preferred for each graphical object 100.

21 The object descriptors 212 are typically stored within the display memory 470 as  
22 a collection of display lists. In the preferred embodiment, each display list corresponds to  
23 a tile. The descriptors for objects (or primitives) that overlap multiple tiles are placed in  
24 more than one display list, each list is sorted in order of depth, and the object descriptors  
25 212 are sorted in tile and depth order. In one embodiment, display list sorting to provide  
26 tile and depth ordering is conducted by the transform engine 460. Tile and depth ordering  
27

1 is preferred to increase efficiency, but is not required. Collectively, the object descriptors  
2 212 describe a graphical scene such as the graphical scene 150.

3 Referring again to Figure 5, the occlusion detector 520 receives a pixel set  
4 descriptor 514, including depth information, and provides a pixel set mask 522. In one  
5 embodiment, the pixel set descriptor describes a horizontal span of consecutive pixels.  
6 The pixel set mask 522 preferably comprises one bit per pixel location within the pixel  
7 set defined by the pixel set descriptor 514. The pixel set mask 522 indicates which pixels  
8 within the pixel set are potentially visible or alternately, which pixels locations were  
9 previously rendered at a shallower depth, and therefore need not be rendered.

10 The pixel generators 510 coordinate with the occlusion detectors 520 to prune or  
11 gate pixels that are known to be occluded and in response provide the potentially visible  
12 pixels 512. Conducting gated pixelization, via the occlusion detectors 520, reduces the  
13 processing and storage burden on the graphics engine 480, particularly the pixel  
14 generators 510, and reduces the required size of the sorting z-buffer 530.

15 The sorting z-buffer 530 receives the potentially visible pixels from the pixel  
16 generators 510. The sorting z-buffer 530 sorts the potentially visible pixels into regions  
17 to facilitate using a relatively small z-buffer referred to as a region-sized z-buffer 545.  
18 The sorted pixels are processed one region at a time, by the region-sized z-buffer 545 to  
19 provide visible pixels 532. In certain embodiments, where pixel transparency is  
20 supported, multiple pixel descriptors for the same pixel location are provided to the  
21 colorization engine 550.

22 The colorization engine 550 colorizes the visible pixels 532 to provide colorized  
23 pixels 552. Colorizing the pixels may involve a wide variety of operations that effect the  
24 final rendered color of each pixel. In one embodiment, colorizing the pixels includes  
25 operations selected from texturing, shading, environmental reflectance mapping, and  
26 shadowing.  
27

1 The colorized pixels 552 are filtered by an anti-aliasing filter 570 to provide the  
2 rendered pixels 482. The graphics engine 480 also includes a pixel attribute memory 580  
3 containing information such as texture maps, color tables, and the like. The information  
4 within the pixel attribute memory 580 is used by the colorization engine 550 to conduct  
5 colorizing operations.

6 As depicted in Figure 5, the sorting z-buffer 530 includes a region sorter 535, a  
7 region memory 540, and a region-sized z-buffer 545. The region sorter 535 receives the  
8 potentially visible pixels 512 and groups the pixels into regions based on their x,y  
9 coordinates within the graphical scene 150. In one embodiment, the region sorter 535 is a  
10 bucket sorter that uses selected high order bits of the x and y coordinates as a sorting key  
11 to sort the potentially visible pixels 512.

12 In the depicted embodiment, the potentially visible pixels 512 are distributed into  
13 the region memory 540 via a memory bus 542 to locations that correspond to specific  
14 regions within the graphical scene 150. In one embodiment the region memory locations  
15 are dynamically allocated to specific regions and are accessed via a linked list. The  
16 sorted pixels 537 corresponding to a region within the graphical scene 150 are removed  
17 from the region memory 540 by the region sorter 535 and are processed by the region-  
18 sized z-buffer 545 to provide the visible pixels 532.

19 Sorting the pixels into regions facilitates the use of a very small z-buffer. The  
20 screen regions corresponding to the region-sized z-buffer 545 are preferably smaller than,  
21 and aligned with, the tiles 310. In one embodiment, multiple pass hyper-sorting is  
22 conducted such that each region is a single pixel and the region-sized z-buffer 545 is  
23 essentially a register.

24 Sorting the pixels into regions also facilitates the use of larger tiles within a  
25 rendering system. Larger tiles reduce the processing load on the graphics engine 480, as  
26 a greater fraction of the primitives comprising the graphical objects 100 are contained  
27



1 within a single graphical tile 310. In one embodiment, the tile 310 is equivalent to the  
2 screen 300.

3 The region-sized z-buffer 545 preferably stores a pixel for each x, y position  
4 within a region of the graphical scene 150. A pixel is overwritten only if it has a pixel  
5 depth that is less than the depth of the currently stored pixel. After processing all of the  
6 sorted pixels 537 corresponding to a region, the pixels remaining within the region-sized  
7 z-buffer 545 are presented as the visible pixels 532.

8 The sorting z-buffer 545 facilitates the usage of complex pixel descriptors while  
9 using a relatively small local memory. Another benefit of the sorting z-buffer 545 is the  
10 ability to conduct deferred shading and texturing while significantly reducing external  
11 memory accesses. The sorting z-buffer 545 also minimizes the processing load on the  
12 rest of the graphics pipeline 480, particularly the colorization engine 550.

13 The colorization engine 550 depicted in Figure 5 includes a set of pixel colorizers  
14 555, an attribute request sorter 560, and a set of attribute request queues 565. The pixel  
15 colorizers 555 receive the visible pixels 532 including descriptive information used to  
16 colorize the pixels. The descriptive information is used to generate attribute requests 557  
17 that are sent to the attribute request sorter 560.

18 The attribute request sorter 560 sorts and directs the attribute requests 557 to the  
19 attribute request queues 565. In one embodiment, the attribute request sorter sorts the  
20 attribute requests 557 according to the memory page in which the requested attribute is  
21 stored. The attribute request sorter 560 also directs the sorted requests to provide one or  
22 more sorted attribute requests 562 to the pixel attribute memory 580. The pixel attribute  
23 memory 580 receives the sorted attribute requests 562 and provides one or more pixels  
24 attributes 582.

25 Sorting the attribute requests increases the effective bandwidth to external storage  
26 by increasing the locality of memory references. This facilitates the use of a larger  
27 amount of slower, lower cost memory with the same effective bandwidth as faster

1 memory, or greater texture storage bandwidth with the same memory technology. It  
2 allows complex multiple lookup texturing and shading algorithms to be conducted  
3 efficiently by repeatedly calculating the address of the next item data to be looked up then  
4 looking them all up in batches between sorting steps.

5 The pixel attributes 582 are received by the pixel colorizers 555 and are used to  
6 colorize the visible pixels 532. Colorizing only visible pixels reduces the processing load  
7 on the graphics engine 480. In one embodiment, colorization comprises shading,  
8 texturing including surface normal perturbation, as well as bi-directional reflectance data  
9 lookup for shading.

10 The various mechanisms of the graphics engine 480 work together to accomplish  
11 high performance rendering using reduced system resources. In certain embodiments, the  
12 reduced usage of resources facilitates the super-sampling of pixels, which is preferred  
13 when rendering voxel objects. Super-sampling involves rendering at a resolution that is  
14 too detailed to be displayed by the output device, followed by filtering and down-  
15 sampling to a lower resolution image that is displayable by the output device.

16 For example, in one embodiment, super-sampling involves generating a 3x3 grid  
17 of super-sampled pixels for each pixel displayed. The 3x3 grid of super-sampled pixels  
18 are low-pass filtered and down-sampled by the anti-aliasing filter 570 to provide the  
19 rendered pixels 482. Super-sampling increases image quality but also significantly  
20 increases the processing and storage requirements of graphical systems.

21 Referring to Figure 6, one embodiment of a graphical rendering method 600 may  
22 be conducted independently of, or in conjunction with, the graphics engine 480. The  
23 graphical rendering method 600 may be conducted in hardware, software, or a  
24 combination of the two. The graphical rendering method 600 commences with a start  
25 step 610 followed by a generate step 620. The generate step 620 provides potentially  
26 visible pixels from a descriptor such as the object descriptor 212.  
27

1 The graphical rendering method 600 proceeds from the generate step 620 to a sort  
2 step 630. The sort step 630 sorts pixels such as the potentially visible pixels 512 into a  
3 plurality of screen regions. In one embodiment, the sort step 630 sorts using the most  
4 significant bits of each pixel's x,y coordinates.

5 The sort step 630 is followed by a z-buffer region step 640. The z-buffer region  
6 step 640 may be conducted in conjunction with the region-sized z-buffer 545. The z-  
7 buffer region step 640 retains the pixel with the shallowest depth for each unique x,y  
8 coordinate in a screen region. If transparency is being used, more than one pixel per x,y,  
9 coordinate may be retained and sent on to the colorizing engine. The level of  
10 transparency for each pixel is preferably known at this point. The z-buffer region step  
11 640 is preferably repeated for each screen region referenced in the sort step 630.

12 After the z-buffer region step 640, the graphical rendering method 600 proceeds to  
13 a sort step 650. Attribute requests are calculated based on the memory location of the  
14 texture or other information required to determine the color of each pixel. The sort step  
15 650 sorts multiple attribute requests to increase the locality of memory references, which  
16 maximizes the rate at which data is transferred from internal or external memory by  
17 minimizing the number of new DRAM page accesses. The sort step 650 is followed by a  
18 retrieve step 660, which retrieves the requested pixel attributes.

19 The retrieve step 660 is followed by a colorize step 670 and a filter step 680. The  
20 colorize step 670 uses the pixel attributes to color, texture, and shade pixels to provide  
21 colorized pixels. The filter step 680 removes aliasing effects by filtering the colorized  
22 pixels. The graphical rendering method 600 terminates at an end step 690.

23 As mentioned, the graphical rendering method 600 may be conducted in  
24 conjunction with the graphics engine 480. Specifically, the generate step 620 is  
25 preferably conducted by the pixel generators 510 and the occlusion detectors 520. The  
26 sort step 630 and the z-buffer region step 640 are preferably conducted in conjunction  
27 with the sorting z-buffer 530. The sort step 650, the retrieve step 660 and the colorize

step 670 are in one embodiment conducted in conjunction with the colorization engine 550 and the pixel attribute memory 580. Lastly, the filter step 680 is preferably conducted in conjunction with the anti-aliasing filter 570.

Figure 7 is a schematic block diagram depicting one embodiment of the pixel generators 510 of Figure 5. As depicted, the pixel generators 510 include a plurality of patch tessilators 710, triangle pixelizers 720, and voxel ray tracers 730. The pixel generators 510 receive the object descriptors 212, and coordinate with the occlusion detectors 520 via an occlusion bus 702, to generate the potentially visible pixels 512.

In one embodiment, the object descriptors 212 received by the patch tessilator 710 describe surface patches such as bezier patches. The patch tessilator 710 converts the surface patches into triangle descriptors 712. The triangle pixelizers 720 receive the triangle descriptors 712 from the patch tessilator 710 or the object descriptors 212 that describe triangles from a module such as the display memory 210. The triangle pixelizers 720 in turn provide the potentially visible pixels 512.

The voxel ray tracers 730 receive the object descriptors 212 that describe or reference voxel objects. Voxel objects are essentially three-dimensional bitmaps that may include surface normal information for each voxel. The voxel ray tracers 730 conduct ray tracing operations that sample voxel objects to provide the potentially visible pixels 512.

The patch tessilators 710 and the triangle pixelizers 720 are exemplary of the architecture of the pixel generators 510. Pixelizers such as the triangle pixelizers 720 receive primitive objects and convert the objects to pixels. The voxel ray tracer 730 is also a pixelizer in that voxels are primitive objects, and the voxel ray tracer 730 provides potentially visible pixels 512. In contrast to pixelizers, converters such as the patch tessilators 710 receive non-primitive objects and convert them to primitive objects that are then processed by pixelizers. Other types of converters and pixelizers may be used within the pixel generators 510.

Table 1 depicts one embodiment of a pixel descriptor used in conjunction with certain embodiments of the present invention. The pixel descriptor may be dependent on the particular type of graphical object 100 that is being processed. For instance, pixel descriptors containing data corresponding to patch objects may differ in structure from pixel descriptors containing data corresponding to voxel objects.

In certain embodiments, the various elements of the graphics engine 480 and the graphical rendering method 600 reference or provide information to the pixel descriptor. For example, in the preferred embodiment, the pixel generators 510 may provide the X,Y location of the pixel within the tile, the Z depth value, the I.D. of the object that generated it, the U,V texture coordinates, and the nX,nY,nZ surface normal values, while the pixel colorizers 555 provide the R, G, and B values. Pixels generated from voxel objects may not utilize all of the fields, such as the surface normal information that may be looked up after the z-buffering stage. The pixel descriptor is preferably dynamic in that fields are added or deleted as required by the stage of the pipeline working with it.

Pixel Descriptor					
R, G, B	Color Index	X, Y, Z	U,V	nX, nY, nZ	Object ID

Table 1

In one embodiment, the pixel descriptor is used to represent the potentially visible pixels 512, the visible pixels 532, and colorized pixels 552. Using a pixel descriptor facilitates a decentralized architecture for the graphics engine 480, such as the flow-thru architecture described in conjunction with Figure 5. The pixel descriptor shown in Table 1 includes values for the device component colors such as the Red, Green, and Blue color values shown in conjunction with the rendered color 108 depicted in Figure 1a. Also included are a color index for the object color, the X, Y, and Z coordinates for the

particular pixel, a pair of texture map coordinates U, V, and surface normal information nX, nY, and nZ.

Referring to Figure 8a, one embodiment of the triangle pixelizer 720 includes a span generator 810 and a span converter 820. The span generator 810 receives the triangle descriptors 712 or the object descriptors 212 that describe triangles and provides a set of spans 812 that are enclosed by the described triangles. In certain situations, the span generator 810 may not generate any of the spans 812. For example, a triangle on its edge may be too thin, and some triangles may be too small to enclose any spans 812.

In the depicted embodiment, the span generator 810 provides a pixel set descriptor 514 to the occlusion detector 520. In return, the occlusion detector 520 provides the pixel set mask 522 indicating which pixels within the pixel set are potentially visible. In one embodiment, the span generator 810 ensures, via the occlusion detector 520, that the spans 812 are pixel spans in which no pixels are known to be occluded. If not, the span generator 810 may restrict or subdivide the spans 812, such that no pixels therein are known to be occluded. The span converter 820 receives the spans 812 and converts the spans into individual pixels, *i.e.*, the potentially visible pixels 512.

Figure 8b is a flow chart diagram depicting one embodiment of a triangle pixelization method 830 of the present invention. The triangle pixelization method 830 includes a start step 835, a generate spans step 840, a pixelize spans step 850, and an end step 855. The generate spans step 840 converts the object descriptor 212 into the spans 812. In one embodiment, the spans 812 containing pixels that are known to be occluded may be subdivided into spans 812 in which no pixels are known to be occluded.

The pixelize spans step 850 converts the spans 812 into individual pixels to provide the potentially visible pixels 512. The triangle pixelization method 830 may be appropriate for objects other than triangles. The triangle pixelization method 830 may be conducted independently of, or in conjunction with, the triangle pixelizer 720.

1 Figure 8c depicts the results typical of the triangle pixelization method 830. An  
2 object boundary 860 is defined by connecting a set of object vertices 862. The object  
3 boundary 860 encompasses a set of pixels 864 that are within the object boundary. The  
4 generate spans step 840 converts the object descriptor 212 into the spans 812. For  
5 example, spans may be computed using geometric formulas that calculate the minimum  
6 and maximum x values for each pixel scanline using slope information. The minimum  
7 and maximum x values correspond to a start pixel and an end pixel of the span 812.

8 Referring now to Figure 9, one embodiment of a ray tracing apparatus 900  
9 includes a bundle caster 910, a proximity tester 920, a ray caster 930, and a collision  
10 tester 940. The ray tracing apparatus 900 may be used to embody the voxel ray tracers  
11 730 of Figure 7. The bundle caster 910 receives the object descriptor 212 and provides  
12 one or more proximate rays 912. The ray caster 930 receives the proximate rays 912 and  
13 provides the potentially visible pixels 512.

14 The bundle caster 910 recursively advances a position 914 of a ray bundle. The  
15 proximity tester 920 receives the position 914 and returns a hit signal 922 if the position  
16 914 is proximate to an object of interest or a portion thereof, such as individual voxels.  
17 In one embodiment, the object of interest is a voxel object, the position 914 advances a  
18 distance that corresponds to a proximity distance used by the proximity tester 920, and  
19 the recursive advancement of the position 914 terminates upon assertion of the hit signal  
20 922. The ray bundle that is advanced by the bundle caster corresponds to a screen area or  
21 region within the graphical scene 150.

22 In the depicted embodiment, the bundle caster provides an individual ray 912 to  
23 the ray caster 930. The ray caster 930 recursively advances a position 932 of an  
24 individual ray. The collision tester 940 receives the position 932 and returns a hit signal  
25 942 if the position 932 impinges upon an object of interest. In one embodiment, the  
26 object of interest is a voxel object, and the recursive advancement of the position 932  
27 terminates upon assertion of the hit signal 942.

1 In the depicted embodiment, the bundle caster 910 and the ray caster 930  
2 communicate with the occlusion detector 520 via the occlusion bus 702 which in one  
3 embodiment carries the pixel set descriptor 514 and the pixel set mask 522. The position  
4 914 that is advanced by the bundle caster 910 and the position 932 that is advanced by the  
5 ray caster 930 each have a depth component that corresponds to a pixel depth within the  
6 graphical scene 150.

7 The bundle caster 910 and the ray caster 930 provide information to one or more  
8 occlusion detectors sufficient to ascertain which rays have a pixel depth greater than the  
9 current occlusion depth. The pixels that are potentially visible are provided by the ray  
10 caster 930 as the potentially visible pixels 512.

11 In one embodiment, the ray caster 930 informs the occlusion detector 520 via the  
12 occlusion bus 702 regarding the depth at which occlusion occurs, *i.e.*, the depth at which  
13 an object of interest is impinged. In the preferred embodiment, the occlusion detector  
14 520 uses the depth information to ascertain the occluded pixels and to update the current  
15 occlusion depth for each pixel position within the pixel set.

16 Referring to Figure 10a, one embodiment of the proximity tester 920 includes a  
17 mask index calculator 1010, a proximity mask cache 1020, and an external memory 1030.  
18 The caching architecture of the proximity tester 920 reduces the required size of local  
19 storage such as on-chip memory. The caching architecture also allows facilitates the use  
20 of slower non-local memory, such as off-chip memory, and lowers the access bandwidth  
21 required of the non-local memory since only the data likely to be used need be brought  
22 on-chip.

23 The mask index calculator 1010 receives the position 914 and computes an index  
24 1012 corresponding to the position 914. The proximity mask cache 1020 contains bit  
25 fields indicating the positions that are proximate or within an object of interest. The  
26 indexed mask bit is preferably within the proximity mask cache 1020 and is used to  
27 provide the hit signal 922. If the mask bit corresponding to the index 1012 is not within



1 the proximity mask cache 1020, the proper mask bit is retrieved via the external memory  
2 1030.

3 Referring to Figure 10b, one embodiment of a collision tester 940 includes a  
4 subblock index calculator 1040, a subblock register 1050, a subblock cache 1060, and an  
5 external memory 1070. The collision tester 940 partitions collision bits indicating the  
6 positions in rendering space that an object of interest occupies into three-dimensional  
7 subblocks such as a 4x4x4 grid of collision bits.

8 To increase the hit rate within the subblock cache 1060 and to facilitate efficient  
9 memory transfers, the various functional units of the collision tester 940 operate on a  
10 subblock basis using a subblock 1062. The use of subblocks and a subblock cache  
11 within the collision tester 940 facilitates the use of slower non-local memory, such as off-  
12 chip memory, and lowers the access bandwidth required of the non-local memory.  
13 Subblocks also reduce the required size of local storage such as on-chip memory. In the  
14 preferred embodiment, the use of subblocks and the subblock cache 1060 within the  
15 collision tester 940 allows the mask tests to be conducted very quickly since the subblock  
16 in use is stored locally to the ray caster.

17 The subblock index calculator 1040 receives the position 932 and computes a  
18 subblock index 1042 as well as a bit index 1044. The subblock index 1042 is received by  
19 and used to access the subblock cache 1060. If the referenced subblock 1062 is within  
20 the cache, it is provided to the subblock register 1050. If not, the referenced subblock  
21 1062 is retrieved from the external memory 1070 and is provided to the subblock register  
22 1050. The bit index 1044 is used to address specific collision bits within the subblock  
23 register 1050 and to provide the hit signal 942.

24 Referring to Figure 11, one embodiment of a caster 1100 includes a set of register  
25 files 1110 and a set of ALU's 1120 to compute the x, y, z, and depth coordinates of a ray  
26 or ray bundle. The caster 1100 may be used to embody the bundle caster 910 and/or the  
27 ray caster 930. The architecture of the caster 1100 facilitates using a wide variety of

The ray casting method 1200 proceeds from the provide step 1210 to a proximity test 1215. The proximity test 1215 ascertains whether the ray bundle is proximate to an object of interest. In one embodiment, the proximity test comprises accessing a mask array in conjunction with the proximity tester 920 shown in Figure 10a and referenced in Figure 9. In another embodiment, the proximity test comprises accessing a distance array

or grid that indicates the shortest distance from each x,y,z position to the graphical object 100.

If the proximity test 1215 is false, the ray casting method 1200 proceeds to an advance bundle step 1220. The advance bundle step 1220 adds a first casting distance to the ray bundle position. In certain embodiments, the advance bundle step 1220 is followed by an occlusion test 1225, which in one embodiment is conducted by the occlusion detector 520.

The occlusion test 1225 ascertains whether the entire ray bundle is known to be occluded (by other objects.) If so, the ray casting method 1200 terminates at an end step 1230. Otherwise, the method loops to the proximity test 1215. In certain embodiments, for instance when an apparatus has ample casting resources and scarce occlusion testing resources, the occlusion test 1225 is not conducted with every casting loop of the ray casting method 1200.

If the proximity test 1215 is true, the ray casting method 1200 proceeds to a subdivide step 1235. The subdivide step 1235 divides the ray bundle into subbundles and continues by processing each sub-bundle. Subdividing requires computing and adding a horizontal and vertical offset (*i.e.* adding a subbundle offset) to the position of the bundle that is subdivided. Subdividing also requires computing a new directional vector in those instances involving perspective rendering. In the preferred embodiment, computing and adding the horizontal and vertical offset is conducted in conjunction with the scalar multiplier 1130 and the ALU 1120.

In certain embodiments, the subdivide step 1235 retreats or advances the ray bundle a second casting distance to ensure proper proximity testing, facilitate longer casting distances and reduce the average number of proximity tests. In one embodiment, the subdivide step retreats a second casting distance, and the average number of proximity and collision tests per ray intersection on typical data was found to be less than eight.

1 In one embodiment, the subdivide step 1235 comprises activating subdivided or  
2 child bundles while continuing to conduct casting of the current (parent) bundle.  
3 Continuing to conduct casting requires proceeding to the advance bundle step 1220 even  
4 when the proximity test 1215 is true. Continued casting of the parent bundle is useful  
5 when some rays may not collide with the object(s) whose proximity is being tested.  
6 Continued casting facilitates termination of the child bundles (*i.e.* rebundling of the  
7 children into the parent) when the proximity test 1215 is once again false, thus reducing  
8 the required number of proximity tests.

9 The subdivide step 1235 is followed by the single ray test 1240, which ascertains  
10 whether the subdivided bundle contains a single ray. If not, the ray casting method 1200  
11 loops to the proximity test 1215. Otherwise, the method 1200 proceeds to a collision test  
12 1245. The collision test 1245 ascertains whether the individual ray has collided with an  
13 object of interest such as the graphical object 100. In one embodiment, the collision test  
14 comprises accessing a mask array in conjunction with the collision tester 940 shown in  
15 Figure 10a and referenced in Figure 9. If the collision test 1245 is false, the ray casting  
16 method 1200 proceeds to an advance ray step 1250.

17 In one embodiment, the advance ray step 1250 adds a first casting distance to the  
18 individual ray position. In another embodiment, the advance ray step 1250 computes the  
19 distance to the next intersected voxel of a voxel object, and advances that distance. In  
20 certain embodiments, the advance bundle step 1220 is followed by an occlusion test  
21 1255, which in one embodiment is conducted by the occlusion detector 520. In certain  
22 embodiments, the occlusion test 1255 is preferably conducted in conjunction with the  
23 subdivide step 1235.

24 The occlusion test 1255 ascertains whether the individual ray is known to be  
25 occluded (by other objects.) If so, the ray casting method 1200 terminates at an end step  
26 1260, otherwise the method 1200 loops to the collision test 1245. In certain  
27

embodiments, the occlusion test 1255 is not conducted for every loop of the advance ray step 1250.

The best placement and frequency of conducting the occlusion test 1225 and 1255 within the ray casting method 1200 may be application-dependent. In particular, the frequency of testing may be adjusted in response to resource availability such as processing cycles within the occlusion detector 520. In certain embodiments, the occlusion test 1225 and 1255 are preferably conducted in conjunction with the provide step 1210 and the subdivide step 1235 rather than after the advance bundle step 1220 and the advance ray step 1250.

Figure 13a is a flow chart diagram depicting one embodiment of a proximity mask generation method 1300 in accordance with the present invention. The generated proximity mask and associated collision mask are preferably used in conjunction with the ray casting method 1200. Figures 13b through 13g are a series of two-dimensional illustrations depicting examples of the results of the proximity mask generation method 1300. The illustrations are presented to enable one of ordinary skill in the art to make and use the invention.

The graphical object 100 shown in Figure 13b may be a voxel object comprised of three-dimensional cubes or voxels. For simplicity, a profile view was selected to restrict the illustration to two dimensions. A voxel object is essentially a three-dimensional bitmap wherein each cell or cube is assigned a color or texture along with a surface normal to indicate the directionality of the surface.

After starting 1310, the proximity mask generation method 1300 proceeds by converting 1320 the graphical object 100 to a collision mask 1322 at the highest resolution available. Converting a voxel object to a collision mask involves storing a single bit for each voxel or cell, preferably in a compressed format.

After creating the collision mask 1322, the proximity mask generation method 1300 proceeds by horizontal copying 1330 the collision mask 1322 in each horizontal

During the advancement of the ray bundles and individual rays, occlusion tests may be conducted to ascertain whether the object of interest is occluded by other graphical objects at the current position of the ray bundle or individual ray. A parent bundle 1410 with an initial position 1412 is tested against a first proximity mask 1420. The proximity test is false resulting in the parent bundle 1410 being cast a first casting distance 1430. The first casting distance 1430 preferably corresponds with the resolution of the first proximity mask 1420 such that visible objects will not be skipped.

1 In the depicted operation, the parent bundle 1410 advances to a second position  
2 1414, whereupon another proximity test is conducted. The proximity test at the second  
3 position 1414 yields a false result, causing the parent bundle 1410 to advance to a third  
4 position 1416. As depicted, the proximity test at the third position 1416 is true, resulting  
5 in sub-dividing of the parent bundle 1410 into child bundles 1440.

6 In the depicted operation, the process of testing and subdividing is repeated for a  
7 second proximity mask 1422 using a second casting distance 1432, a third proximity  
8 mask 1424 using a third casting distance, and so forth, until the bundles are subdivided  
9 into individual rays. The individual rays are then tested against a collision mask 1450  
10 where a true result indicates impingement upon a potentially visible object. During the  
11 advancement of the ray bundles and individual rays, occlusion tests may be conducted to  
12 ascertain whether the object of interest is occluded by other graphical objects at the  
13 current position of the ray bundle or individual ray.

14 Figures 15, and 16 are illustrations depicting the operation of the ray casting  
15 method 1200 of the present invention. Referring to Figure 15a, a ray bundle 1510  
16 comprises individual rays 1511 and occupies a volume 1512 in rendering space. In the  
17 depicted embodiment, the volume 1512 is a cube with a width 1514, a height 1516, and a  
18 length 1518. An object of interest 1520 is subject to proximity tests of various distances.  
19 Successful casting requires choosing a selected proximity 1530, which ensures that the  
20 object of interest 1520 is not skipped when within the graphical scene 150, and that a  
21 casting distance 1535 is not unnecessarily short. In one embodiment, the selected  
22 proximity 1530 corresponds to an enlarged object of interest 1520a.

23 Proper proximity testing requires that the selected proximity 1530, i.e., the  
24 amount of enlargement used in creating a proximity mask, is greater than a distance 1540  
25 from a testing position 1550 to the furthest point within the volume 1512. The selected  
26 proximity 1530 must therefore be greater than or equal to the distance 1540, and the  
27 testing position 1550 is preferably in the center of the volume 1512.

Referring to Figure 16, a ray bundle 1610 may be comprised of diverging rays 1612 that originate from the focal point 114 of the perspective viewer 106 shown in Figure 1a. With diverging rays, the volume 1512 increases with each successive cast due to the increase in width 1514 and height 1516. In one embodiment, proper proximity testing is maintained by recalculating the distance 1540 and selecting a proximity mask with an object enlargement that is greater than or equal to the distance 1540.

Referring to Figure 17a, one embodiment of the occlusion detector 520 of Figure 5 includes a coarse z-buffer 1710, a comparator 1720, and a register 1730. The coarse z-buffer 1710 is in one embodiment essentially a specialized memory containing the shallowest known pixel depth for each pixel position in the graphical scene 150. The shallowest known depth is the shallowest depth encountered at each pixel position for the pixels that have already been processed by the occlusion detector 520. The shallowest known pixel depth is referred to herein as the current occlusion depth.

The data bus 1712 carries the depth information that is stored within the coarse z-buffer. In one embodiment, the data bus 1712 is a parallel bus that is capable of accessing an entire row of depth information within the coarse z-buffer 1710. In another embodiment, the data bus 1712 (and the pixel set mask 522) is a convenient width such as 32 bits and multiple accesses must be conducted to access an entire row of depth information. The entire row of depth information preferably corresponds to a row of pixels within the graphical scene 150. The depth information is preferably coarse, i.e., of a reduced resolution in that complete pixel pruning is not required by the occlusion detector 520.

Using coarse depth information (*i.e.*, a reduced number of bits to represent the depth) facilitates pruning the majority of occluded pixels while using a relatively small memory as the coarse z-buffer 1710. In one embodiment, the coarse z-buffer 1710 is used in conjunction with depth shifting in which graphical rendering is localized to a



1 specific depth range and the display lists are sorted in depth (front-to-back) order to  
2 facilitate depth localization.

3 Depth shifting or depth localization is a method developed in conjunction with the  
4 present invention to maximize the usefulness of the coarse z-buffer. Depth shifting  
5 comprises shifting a depth range during the rendering process thereby focusing the  
6 resolution of the coarse z-buffer to a particular range of z values. In the preferred  
7 embodiment, a current minimum depth is maintained along with a current coarseness, for  
8 example, a multiplier or exponent, indicating the resolution of the z values stored within  
9 the coarse z-buffer. Depth shifting is preferably conducted in conjunction with depth  
10 ordered rendering, and the current coarseness is adjusted to match the density of  
11 primitives being rendered at the current depth.

12 In one embodiment, depth shifting comprises subtracting an offset from each z  
13 value within the z-buffer, with values below zero being set to zero. In another  
14 embodiment, depth shifting comprises subtracting an offset as well as bit shifting each of  
15 the z values to change the current coarseness of values contained in the coarse z-buffer.  
16 In yet another embodiment, depth shifting comprises adding an offset to the values in the  
17 coarse z-buffer and setting overflowed depths to a maximum value and underflowed  
18 depths to a minimum value. In the presently preferred embodiment, the maximum z  
19 value represented in the coarse z-buffer indicates a location containing no pixel data,  
20 while the minimum value of zero represents a pixel generated at a shallower depth than  
21 the current minimum depth.

22 The register 1730 receives a pixel set descriptor 514 including depth information.  
23 In one embodiment, the pixel set descriptor 514 describes a horizontal span of  
24 consecutive pixels. The register 1730 provides the pixel set descriptor to the comparator  
25 1720.

26 The comparator 1720 compares the minimum depth for the pixel set with each  
27 pixel's occlusion depth by accessing the occlusion depth for each pixel within the pixel

1 set via the data bus 1712. The comparator 1720 provides the pixel set mask 522  
2 indicating which pixels within the pixel set are known to be occluded. In the preferred  
3 embodiment, the comparator 1720 also compares the maximum depth for the pixel set  
4 with each pixel's occlusion depth and updates the contents of the z-buffer if the  
5 maximum depth is shallower than the current occlusion depth.

6 Referring to Figure 17b, one embodiment of an occlusion detection method 1740  
7 may be conducted in conjunction with the generate step 620 of the graphical rendering  
8 method 600 of the present invention. The occlusion detection method 1740 may also be  
9 conducted in conjunction with the occlusion detector 520. In the preferred embodiment,  
10 the occlusion detection method 1740 is used to conduct gated pixelization such that pixels  
11 that are known to be occluded are not included in subsequent rendering stages.

12 The occlusion detection method 1740 begins with a start step 1750 followed by a  
13 receive step 1755. The receive step 1755 receives a pixel set descriptor, such as the pixel  
14 set descriptor 514, that describes the extents of the pixel set being processed in  
15 conjunction with a graphical object such as the graphical object 100. The pixel set  
16 descriptor preferably includes depth information such as maximum and minimum depth.  
17 In one embodiment, the pixel set descriptor enumerates the starting and ending pixels of a  
18 span along with minimum and maximum depths.

19 The occlusion detection method 1740 facilitates specifying a depth range rather  
20 than requiring exact depth information for each pixel in the pixel set of interest. In most  
21 cases, a depth range comprising minimum and maximum depths is sufficient to prune a  
22 majority of non-visible pixels and update the occlusion depth. While the occlusion  
23 detection method 1740 may be used in a single pixel mode that specifies an exact pixel  
24 depth, the preferred embodiment comprises specifying a depth range for an entire set of  
25 pixels. Specifying a depth range for an entire set of pixels reduces the data bandwidth  
26 required to conduct occlusion detection.  
27

1 The occlusion detection method 1740 proceeds from the receive step 1755 to a  
2 retrieve step 1760. The retrieve step 1760 retrieves the occlusion depth for the locations  
3 described by the pixel set descriptor. In one embodiment, the retrieve step 1760 is  
4 conducted by the comparator 1720 in conjunction with the coarse z-buffer 1710.

5 After the receive step 1755, the occlusion detection method 1740 conducts a  
6 minimum depth test 1770 on each pixel in the described pixel set. The minimum depth  
7 test 1770 ascertains whether the occlusion depth for a particular pixel location is less than  
8 the pixel set minimum. If so, the set flag step 1775 is conducted. Otherwise, a maximum  
9 depth test 1780 is conducted. The set flag step 1775 sets a flag for each pixel that passes  
10 the minimum depth test 1770. The pixels that pass the minimum depth test 1770 are  
11 known to be occluded, while the remaining pixels are potentially visible.

12 If the minimum depth test 1770 is false for some or all of the pixels in the pixel  
13 set of interest, the maximum depth test 1780 is conducted preferably only on those pixels  
14 that fail the minimum depth test 1770. The maximum depth test 1780 ascertains whether  
15 the occlusion depth for a particular pixel location is greater than the pixel set maximum.  
16 If so, the particular pixel is shallower than the occlusion depth and an update step 1785 is  
17 conducted to update the occlusion depth.

18 The maximum depth test 1780 and the update step 1785 ensure that the occlusion  
19 depth is only decreased and will not be increased while processing a graphical scene or  
20 frame. Successful occlusion depth updates are contingent on the maximum depth being  
21 valid for the entire set of pixels being considered. In those situations where it is not  
22 known if the graphical object occludes the entire set, such as certain embodiments of the  
23 ray casting method 1200, occlusion depth updates may be deferred until an actual ray  
24 collision occurs thereby removing uncertainty and possible erroneous updates. After the  
25 update step 1785, the occlusion detection method 1740 then loops to the receive step  
26 1755 to process other objects and pixel sets.  
27

The bucket buffers 1850 are essentially cache memory for the memory array 1810 that is under intelligent control of the bucket controller 1860. The bucket controller 1860 orchestrates the movement of data within the bucket sorter 1800 to effect sorting operations. The architecture of the bucket sorter 1800 facilitates sorting data that is already within the memory array 1810. In certain embodiments, multiple sorting passes may be conducted on data within the memory array 1810. In one embodiment, one or more of the bucket write buffers 1850a is a miscellaneous bucket that is resorted after the initial sort. The bucket controller 1860 receives and provides bucket data externally through a set of bucket ports 1862 that, in the depicted embodiment, are partitioned into bucket write ports 1862a and bucket read ports 1862b.

1 In one embodiment, the bucket controller 1860 assigns bucket ID's to each bucket  
2 buffer and transfers filled bucket write buffers 1850a to the memory array 1810 via a  
3 memory buffer 1830 and fills empty bucket read buffers 1850b in like fashion. The  
4 memory bus 1840, the memory buffer 1830, the column bus 1822, and the array columns  
5 1820 are preferably wide enough to transfer an entire bucket buffer in one bus cycle.

6 The bucket controller 1860 is preferably equipped with a mechanism to track the  
7 placement of bucket data within the memory array 1810. In one embodiment, the  
8 tracking mechanism references a memory assignment table, while in another embodiment  
9 the tracking mechanism manages a set of linked lists. The bucket controller 1860 may  
10 dedicate particular bucket buffers 1850 to store tracking data. The bucket controller 1860  
11 may also store tracking data within the memory array 1810. The components of the  
12 bucket sorter 1800 may be partitioned into a memory 1800a and a sorter 1800b.

13 Figure 18b shows additional detail of specific elements related to an on-chip  
14 embodiment of the bucket sorter 1800. The depicted embodiment is configured to utilize  
15 embedded DRAM using wide data paths to increase available bandwidth and bucket  
16 sorting performance. In the depicted embodiment, each memory buffer 1830 includes  
17 multiple sense amps 1830a, one or more transfer registers 1830b, and a data selector  
18 1830c. In one embodiment, the selectors comprise an multiplexor.

19 The depicted bucket buffers 1850 comprise an N bit interface to a bucket bus  
20 1852 and an MxN bit interface to the memory bus 1840. In the depicted embodiment,  
21 each of the K bucket buffers 1850 may transfer data to and from the bi-directional  
22 memory bus 1840. In the preferred embodiment, the bits of the bucket buffer are  
23 interleaved to facilitate bit alignment and to reduce wiring complexity. For example,  
24 with a bucket buffer of M locations of N bit words, the bits of the bucket buffer are  
25 arranged such that the bit cells of the least significant bits from each of the M memory  
26 locations are located on one end of the bucket buffer, while the bit cells of the most  
27 significant bits are located on the other end of the bucket buffer. Such an arrangement

Bucket buffers such as the bucket buffers 1850 may also be assigned to buckets, although in certain embodiments there are fewer bucket buffers than actual buckets. In these embodiments, some bucket buffers may be assigned to a “miscellaneous” or “other” bucket whose contents must be resorted when additional bucket buffers are available.

1 Sorting may also be conducted recursively by dividing available bucket buffers into  
2 groups for example by sorting on a sorting key one bit at a time.

3 The bucket sorting method 1900 proceeds from the allocate step 1920 to a route  
4 step 1930. The route step 1930 writes a data element within the bucket write buffer  
5 1850a that corresponds to a data key. The data element may be received via one of the  
6 bucket write ports 1862a, and for example, may be received from an external functional  
7 or one of the sorter output ports 1852b, such as when recursively sorting data. The data  
8 key may be part of the data element or the data key may be provided separately. After the  
9 route step 1930, the bucket sorting method 1900 proceeds to a buffer full test 1940.

10 The buffer full test 1940 ascertains whether the buffer that was written to is full.  
11 In one embodiment, the buffer full test comprises checking a signal from the particular  
12 bucket write buffer 1850a. If the buffer full test is not true, the bucket sorting method  
13 1900 loops to the route step 1930. Otherwise, the method proceeds to an empty buffer  
14 step 1950.

15 The empty buffer step 1950 transfers the contents of a bucket buffer such as the  
16 bucket buffer 1850 to a region of memory associated with a particular bucket. In certain  
17 embodiments, the empty buffer step 1950 is followed by a bucket full test 1960. The  
18 bucket full test 1960 ascertains whether the region of memory associated with a particular  
19 bucket is full.

20 If the tested bucket is full, the bucket sorting method 1900 loops to the allocate  
21 step 1920 where in one embodiment additional memory is allocated. Otherwise, the  
22 bucket sorting method 1900 loops to the route step 1930 to process additional data  
23 elements. The buffer full test 1940, the empty buffer step 1950, and the bucket full test  
24 1960 are preferably conducted in parallel for each bucket buffer.

25 Referring to Figure 20a, one embodiment of the sorting z-buffer 530 uses the  
26 bucket sorter 1800 to embody the sorting z-buffer 530. Specifically, the region sorter 535  
27

The request sorter 2110 receives an access request 2108, which in one embodiment comprises an address field, a data field, and an operation field. Multiple access requests 2108 are received and sorted into the page access queues 2120 via an



Figure 22 relates the certain elements of the graphics engine with the bucket sorter 1800. A pixel colorizer 2200 includes a set of address calculators 555a, a set of attribute processors 555b, the attribute request sorter 560, the attribute request queues 565, and the pixel attribute memory 580. The address calculators 555a and the attributes processors

1 555b may comprise the pixel colorizers 555 shown in Figure 5, while the pixel colorizer  
2 2200 may be contained within the graphics engine 480.

3 In the depicted embodiment, the pixel colorizer 2200 includes a pixel combiner  
4 2210. The pixel combiner 2210 is preferred in embodiments that conduct super-sampled  
5 rendering. Super-sampled rendering increases visual quality by rendering a set of pixels  
6 for each output pixel. The set of rendered pixels are filtered (*i.e.*, smoothed) to provide  
7 each output pixel.

8 The pixel combiner 2210 examines the visible pixels 532 that comprise a single  
9 output pixel. The pixel descriptors of pixels associated with an output pixel are accessed  
10 to ascertain whether some or all the pixels may be combined into a representative pixel  
11 2212. If not, the visible pixels 532 are passed along without combining them.

12 In one embodiment, combining is performed if multiple pixels originate from the  
13 same patch and texture. In such cases it may not be advantageous to conduct texture  
14 lookups, and shading for all of those subpixels, the associated visible pixels 532 are  
15 discarded from further rendering with the exception of the representative pixel 2212. The  
16 representative pixel 2212 is preferably the center pixel in the set of pixels of the pixels it  
17 represents.

18 In the depicted embodiment, the address calculators 555a compute a memory  
19 address associated with an attribute of interest. The memory address is presented as the  
20 attribute request 557. The attribute request is handled by the request sorter 560 in the  
21 manner related in the description of Figure 5 and provides the sorted attribute requests  
22 562.

23 The attribute processors 555b receive the visible pixels 532 or the representative  
24 pixels 2210 along with the pixel attributes 582 and provide the colorized pixels 552. The  
25 colorized pixels 552 may be recirculated within the pixel colorizer 2200 via a  
26 recirculation bus 2220. Recirculation facilitates the acquisition of additional attributes  
27 for each pixel.

1 Referring to Figure 23, one embodiment of a pixel colorization method 2300 of  
2 the present invention may be conducted independently of, or in conjunction with, the  
3 pixel colorizer 2200 or the graphics engine 480. The pixel colorization method 2300  
4 begins with a start step 2310 followed by a calculate address step 2320, a sort requests  
5 step 2330, and a process queue step 2340.

6 The calculate address step 2320 computes a memory address for a needed  
7 attribute such as a color table entry, a texture map, shading data, and the like. The needed  
8 attributes may be dependent on the type of object from which the pixels originated. The  
9 calculate address step 2320 is preferably conducted for a large number of pixels such as  
10 the visible pixels 532. The pixel colorization method 2300 contributes to the localization  
11 of memory references by processing the same needed attribute for every pixel in the  
12 pixels of interest. Typically, accessing the same attribute focuses the memory references  
13 to a relatively small portion of a graphics memory such as the pixel attribute memory  
14 580.

15 The sort requests step 2330 sorts the preferably large number of the calculated  
16 addresses into page queues to further increase the locality of memory references. The  
17 process queue step 2340 accesses a memory such as the pixel attribute memory 580 with  
18 the sorted addresses. In one embodiment, the process queue step 2340 uses the retrieved  
19 attribute information to colorize the visible pixels 532.

20 The pixel colorization method 2300 proceeds from the process queue to a queues  
21 processed test 2350. The queues processed test 2350 ascertains whether every page  
22 queue with a pending request has been processed. If not, the pixel colorization method  
23 2300 loops to the process queue step 2340. Otherwise, the method proceeds to an  
24 attributes processed test 2360.

25 The attributes processed test 2360 ascertains whether all relevant attributes have  
26 been processed for the pixels of interest such as a frame of visible pixels 532. If not, the  
27

1 pixel colorization method 2300 loops to the calculate address 2320. Otherwise, the pixel  
2 colorization method 2300 terminates at an end step 2370.

3 The present invention may be embodied in other specific forms without departing  
4 from its spirit or essential characteristics. The described embodiments are to be considered  
5 in all respects only as illustrative and not restrictive. The scope of the invention is, therefore,  
6 indicated by the appended claims rather than by the foregoing description. All changes,  
7 which come within the meaning and range of equivalency of the claims, are to be embraced  
8 within their scope.

9 What is claimed is: